Recovering the emergent logic in a software design exercise

Nozomi Ikeya, Department of Humanities and Social Sciences, Keio University, Tokyo, Japan
Rachael Luck, School of Construction Management and Engineering, University of Reading, Reading, UK
Dave Randall, Institute for Information Systems, University of Siegen, 57076 Siegen, Germany

This paper develops a deeper understanding of professional software design by examining the emergent logic of a software design exercise. Decision-making is evident as a 'product' of activity, including coordinated attention to primarily two artefacts, the brief and the whiteboard. Thus, we pay attention to the 'situatedness' of decision-making, which is not one person's accomplishment, but is interactively carried out through treating what is known to the participants such as requirements written in the brief as 'documentary' of what is to be understood. The paper examines how each pair resolved the requirements uncertainties, by treating different 'users' differently. Our examination reveals how different approaches to the design exercise were actually organised to shed new light on software design practices.

Keywords: ethnomethodology, software design, design practice, reasoning, problem solving

It is not surprising to see that qualitative studies of software engineering are becoming increasingly common. They demonstrate a concern for 'situatedness'; the local, practical accomplishment of work as opposed to its formal and managed character. This shift in methodological emphasis was a result of dissatisfaction with existing approaches to enquiries about the design of technology. Critique centred on overly rationalistic assumptions about the design process and the problem of discovering what was 'really' going on rather than what 'ought' to be. In turn, this gave rise to an interest in the contribution that ethnographic studies make to the design process. One of the earliest was Bonnie Nardi's study of end-user programming, in which she pointed out, "there are few in-depth empirical studies of the actual use of end-user programming systems" (Nardi, 1993, p. 9) and that "we are a long way from … a corpus of knowledge that identifies the properties of artifacts and situations that are most significant for design" (Nardi, 1993, p. 58).

Corresponding author:
Nozomi Ikeya
nozomi.ikeya@a8.keio.jp

Founded on similar concerns, there have been a variety of moves in the software engineering process itself. In reaction to models, which arguably had a number of flaws, new and more flexible approaches were demanded. This included a concern for iteration (Boehm, 1988); an emphasis on requirements gathering (Jirotka & Goguen, 1994) and a general dissatisfaction with top-down, highly managed processes. This gave rise to other perspectives on software design, including agile and extreme programming that emphasised rapid responses to customer needs. It seems these problems do not go away. Loucopoulos et al. (2003) note that understanding the application domain 'cannot easily be overestimated … when you have to solve somebody else's problem the first thing you have to do is to find out more about it' … "The 'requirements mess' has remained a pernicious challenge … RE researchers have been persistent to note that the leading sources of project difficulty – lack of user input, incomplete requirements, and changing specifications – are directly related to flaws in design requirements".

There was a 'double' push for change, a push that recognised both the need for software design to be more flexible and responsive, and recognition that new methodologies would be necessary if we were to understand these processes better. The study of collaboration in software development, the activities and work practices of professional software designers has become a burgeoning field (Button and Sharrock, 1995; Hughes et al., 1993; Martin & Rooksby, 2006; Sharrock & Anderson, 1994), which includes studies of pair programming in various settings (Beck, 2000; Mackenzie & Monk, 2004; Nosek, 1998; Rooksby & Ikeya, 2012). Pair programming and its variants, Agile and Xtreme Programming XP methods, set up a situation where programmers are in constant communication, asking and answering questions of each other (Chong et al., 2005) and thus render design processes more accessible for ethnographic work. We do not mean to suggest that the work evidenced below constitutes pair programming, for it does not. Rather, we might think of this an exercise in design work where certain features of the work undertaken in the exercise allow us to make some judgements about how professional software developers, working intensely in pairs, and under strict time pressure, do the work that they do.

1 Pair working in software design

We examine two video transcripts that formed part of the Studying Professional Software Designers Workshop. These videos have been examined before. Baker and van der Hoek (2010) show how the sessions "*were characterized by the repeated discussion and recontextualization of a set of core ideas and subjects. Specifically, as the designs evolved, previously stated ideas were frequently restated and reconsidered. … The result was a process that had characteristics of breadth-first design and incremental design, but that delved into details differently; we found that there was a focus on finding key, organizing ideas and adjusting them to work relative to one another to cre-*